Abstract

Malware authors employ sophisticated tools and infrastructure to undermine information security and steal data on a daily basis. When these attacks or infrastructure are discovered, digital forensics attempts to reconstruct the events from evidence left over on file systems, network drives, and system memory dumps. In the last several years, malware authors have been observed using the Java managed runtimes to commit criminal theft and conduct espionage and spying [1, 2, 3].

Fortunately for forensic analysts, the most prevalent versions of Java use generational garbage collection to help improve runtime performance. The memory system allocates memory from a managed heap. When memory is exhausted in this heap, the JVM will sweep over partitions reclaiming memory from dead objects. This memory is not sanitized or zeroed. Hence, latent secrets and object data persist until it is overwritten [4, 5]. For example, sockets and open file recovery are possible even after resources are closed and purged from the OS kernel memory.

This research measures the lifetime of latent data and implements a Python framework that can be used to recover this object data. Latent secret lifetimes are experimentally measured using TLS keys in a Java application. An application is configured to be very active and minimally active. The application also utilizes raw Java sockets and Apache HTTPClient to determine whether or not a Java framework impacts latent secret lifetimes. Depending on the heap size (512MiB to 16GiB), between 10-40% of the TLS keys are recoverable from the heap, which correlates directly to memory pressure.

This research also exploits properties to identify and recover evidence from the Java heap. The RecOOP framework helps locate all the loaded types, identify the managed Java heaps, and scan for potential objects [6]. The framework then lifts these objects into Python where they can be analyzed further. One key findings include the fact that IO streams for processes started from within Java remained in memory, and the data in these buffers could be used to infer the program executed. Socket and data could also be recovered even when the socket structures were missing from the OS’s kernel memory.

REFERENCES


Problem Statement and Goals

- Managed runtimes using Garbage Collection (GC) are susceptible to disclosure attacks
- Managed runtimes are prevalent in enterprise networks
- Threat actors can hide malware behavior in managed runtimes
- Digital forensics can exploit latent secrets to reconstruct events

Approach

- Find relevant runtime structures
- Reconstruct the loaded types
- Locate managed memory
- Enumerate heaps
- Scan for object signatures
- Interpret the objects and values
- Mine for evidence

Results

- Recovered latent process information, objects, Java bytecode, obfuscated or encrypted classes, etc.
- Reconstructed events based on recovered evidence
- Extend approach to other managed runtimes
- Reduce resource requirements and increase execution performance

Special thanks to Dr. Simson Garfinkel and Dr. Dan Wallach, and this research was supported in part by NSF grants CNS-1409401 and CNS-1314492 and the NPSC Fellowship.